Backpropagation

Deep Learning (DSE316/616)

Vinod K Kurmi
Assistant Professor, DSE

Indian Institute of Science Education and Research Bhopal

Aug 18, 2022

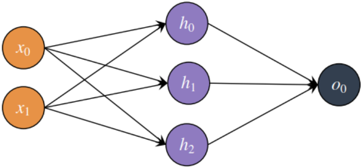
Disclaimer

- Much of the material and slides for this lecture were borrowed from
    - Bernhard Schölkopf's MLSS 2017 lecture,
    - Tommi Jaakkola's 6.867 class,
    - CMP784: Deep Learning Fall 2021 Erkut Erdem Hacettepe University
    - Fei-Fei Li, Andrej Karpathy and Justin Johnson's CS231n class
    - Hongsheng Li's ELEG5491 class

Previous Class: Recap

Input

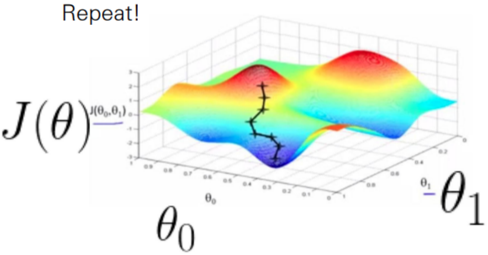
[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted	Actual
0.05	1
0.02	0
0.96	1
0.35	1

$$J(\theta) = \frac{1}{N}\sum_i \ell(\underbrace{f(\mathbf{x}^{(i)};\theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Previous Class: loss surface



Previous Class: SGD

- Initialize θ randomly
- For N Epochs
  - For each training batch $\{(x_0, y_0), (x_1, y_1), ...(x_B, y_B)\}$:
    - Compute Loss Gradient: $\frac{\partial \mathcal{J}(\theta)}{\partial \theta} = \frac{1}{B} \sum_i \frac{\partial \mathcal{J}_i(\theta)}{\partial \theta}$
    - Update θ with update rule: $\theta = \theta - \eta \frac{\partial \mathcal{J}(\theta)}{\partial \theta}$

Advantages

- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Minibatches lead to fast training!
  - Can parallelize computation

Previous Class: Training NN

- $a_L(x;\theta_1,...,\theta_L) = h_L(h_{L-1}(...h_1(x;\theta_1),\theta_{L-1});\theta_L)$
  - x: input
  - θ_l: parameter of layer l
  - $a_l = h_l(x;\theta_l)$: (non)-linear function
- Given training corpus X, Y find optimal parameters

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{(x,y)\in(X,Y)} \ell(y, a_L(x;\theta_1,...,\theta_L))$$

- To use any gradient descent based optimization

$$\theta^{(t+1) = \theta^t - \eta_t \frac{\partial \mathcal{L}}{\partial \theta_t}}$$

- we need the gradients $\frac{\partial \mathcal{L}}{\partial \theta_l}$; $l = 1, 2, ...L$
- How to compute the gradients for such a complicated function enclosing other functions, like $a_l(...)$

Training a neural network

- Given:
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples $S = \{(xi, yi)\}$
- The goal: Learn the weights of the neural network
- **Remember**: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction $y = NN(x; w)$

Training a neural network

- Given:
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples $S = \{(xi, yi)\}$
- The goal: Learn the weights of the neural network
- **Remember**: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction $y = NN(x; w)$

Training a neural network

- Given:
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples $S = \{(xi, yi)\}$
- The goal: Learn the weights of the neural network
- **Remember**: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction $y = NN(x; w)$

Training a neural network

- Given:
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples $S = \{(xi, yi)\}$
- The goal: Learn the weights of the neural network
- **Remember**: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction $y = NN(x; w)$

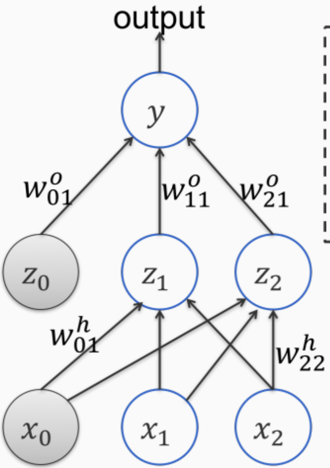
Training a neural network

- Given:
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples $S = \{(xi, yi)\}$
- The goal: Learn the weights of the neural network
- **Remember**: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction $y = NN(x; w)$

Training a neural network

- Given:
  - A network architecture (layout of neurons, their connectivity and activations)
  - A dataset of labeled examples $S = \{(xi, yi)\}$
- The goal: Learn the weights of the neural network
- **Remember**: For a fixed architecture, a neural network is a function parameterized by its weights
  - Prediction $y = NN(x; w)$

Back to our running example



Given an input **x**, how is the output predicted

output $y = w^o_{01} + w^o_{11} z_1 + w^o_{21} z_2$

$$z_2 = \sigma(w^h_{02} + w^h_{12} x_1 + w^h_{22} x_2)$$

$$z_1 = \sigma(w^h_{01} + w^h_{11} x_1 + w^h_{21} x_2)$$

Suppose the true label for this example is a number y_i

We can write the *square loss* for this example as:

$$L = \frac{1}{2}(y - y_i)^2$$

Checkpoints

- If we have a neural network (structure, activations and weights), we can make a prediction for an input

- If we had the true label of the input, then we can define the loss for that example

- If we can take the derivative of the loss with respect to each of the weights, we can take a gradient step in SGD

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: the original linear classifier

- Linear score function: **f=Wx**
  - $x \in \mathcal{R}^D$
  - $W \in \mathcal{R}^{CxD}$
- Two layer neural network: $f = W_2 max(0, W_1 x)$
  - $x \in \mathcal{R}^D$
  - $W_1 \in \mathcal{R}^{HxD}$
  - $W_2 \in \mathcal{R}^{CxH}$
- In practice we will usually add a learnable bias at each layer as well
- Neural Network is a very broad term; these are more accurately called "fully-connected networks" or sometimes "multi-layer perceptrons" (MLP)

Neural networks: why is max operator important?

- The function is called the activation function.
- Q: What if we try to build a neural network without one?
  - $f = W_2 W_1 x$
  - $W_3 = W_2 W_1 \in \mathcal{R}^{CxD}$
  - $f = W_3 x$
- A: We end up with a linear classifier again!

Neural networks: why is max operator important?

- The function is called the activation function.
- Q: What if we try to build a neural network without one?
  - $f = W_2 W_1 x$
  - $W_3 = W_2 W_1 \in \mathcal{R}^{CxD}$
  - $f = W_3 x$
- A: We end up with a linear classifier again!

Neural networks: why is max operator important?

- The function is called the activation function.
- Q: What if we try to build a neural network without one?
  - $f = W_2 W_1 x$
  - $W_3 = W_2 W_1 \in \mathcal{R}^{CxD}$
  - $f = W_3 x$
- A: We end up with a linear classifier again!

Neural networks: why is max operator important?

- The function is called the activation function.
- Q: What if we try to build a neural network without one?
  - $f = W_2 W_1 x$
  - $W_3 = W_2 W_1 \in \mathcal{R}^{CxD}$
  - $f = W_3 x$
- A: We end up with a linear classifier again!

Neural networks: why is max operator important?

- The function is called the activation function.
- Q: What if we try to build a neural network without one?
    - $f = W_2 W_1 x$
    - $W_3 = W_2 W_1 \in \mathcal{R}^{CxD}$
    - $f = W_3 x$
- A: We end up with a linear classifier again!

Neural networks: why is max operator important?

- The function is called the activation function.
- Q: What if we try to build a neural network without one?
  - $f = W_2 W_1 x$
  - $W_3 = W_2 W_1 \in \mathcal{R}^{CxD}$
  - $f = W_3 x$
- A: We end up with a linear classifier again!

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

Problem: How to compute gradients?

- $s = f(x; W_1, W_2) = W_2 \max(0, W_1 x)$ nonlinear score function
- $\mathcal{L}_i = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$ SVM loss
- $R(W) = \sum_k W_k^2$ regularization
- $L = \frac{1}{N} \sum_i^N L_i + \lambda_1 R(W_1) + \lambda_2 R(W_2)$ Total loss
- If we can compute $\frac{\partial \mathcal{L}}{\partial W_1}$ and $\frac{\partial \mathcal{L}}{\partial W_2}$ then we can learn W_1 and W_2.

$$\min_{w} \sum_{i} L(NN(x_i, w), y_i)$$

Stochastic gradient descent

Given a training set $S = \{(\mathbf{x}_i, y_i)\}, \mathbf{x} \in \Re^d$

1. Initialize parameters w
2. For epoch = 1 … T:
   1. Shuffle the training set
   2. For each training example $(\mathbf{x}_i, y_i) \in S$:
      - Treat this example as the entire dataset
      - Compute the gradient of the loss $\nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i)$ using backpropagation
      - Update: $\boldsymbol{w} \leftarrow \boldsymbol{w} - \gamma_t \nabla L(NN(\boldsymbol{x}_i, \boldsymbol{w}), y_i))$
3. Return w

The objective is not convex. Initialization can be important

γ_t: learning rate, many tweaks possible

Lets Calculate

(Bad) Idea: Derive $\nabla_W L$ on paper

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$= \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda \sum_k W_k^2$$

$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2$$

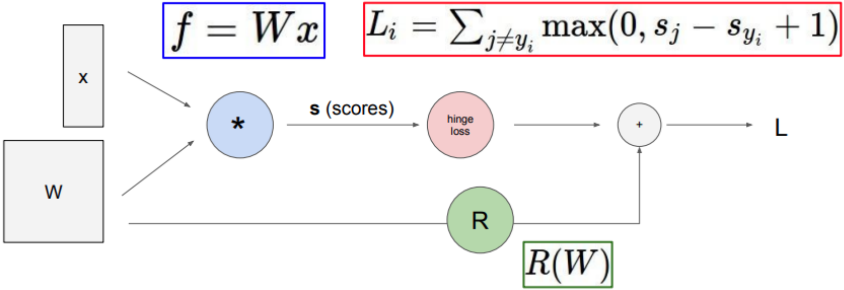
$$\nabla_W L = \nabla_W \left(\frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, W_{j,:} \cdot x + W_{y_i,:} \cdot x + 1) + \lambda \sum_k W_k^2 \right)$$

Problem: Very tedious: Lots of matrix calculus, need lots of paper

Problem: What if we want to change loss? E.g. use softmax instead of SVM? Need to re-derive from scratch =(

Problem: Not feasible for very complex models!

Better Idea: Computational graphs + Backpropagation



Solution: Backpropagation!!!!

Examples

Backpropagation: a simple example

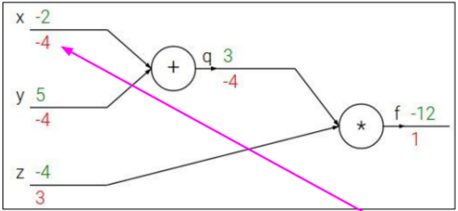
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

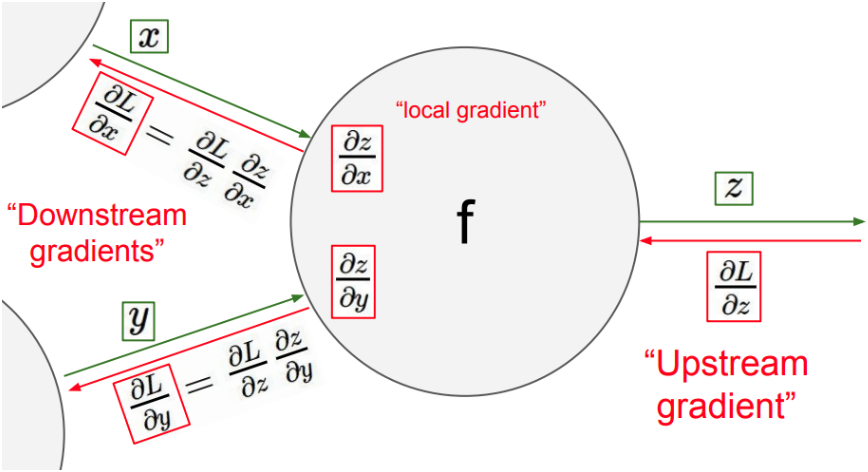
Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Upstream gradient

Local gradient

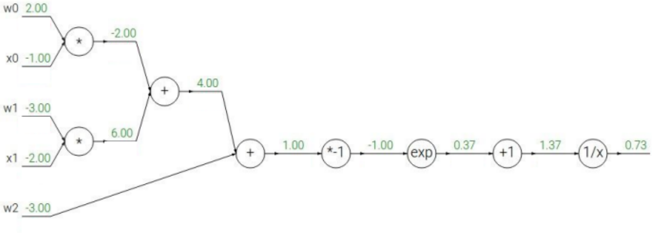
Examples



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"Downstream gradients"
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"

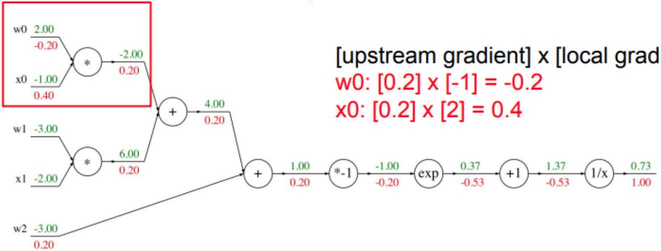
Examples

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



$f(x) = e^x$	$\to$	$\frac{df}{dx} = e^x$	$f(x) = \frac{1}{x}$	$\to$	$\frac{df}{dx} = -1/x^2$
$f_a(x) = ax$	$\to$	$\frac{df}{dx} = a$	$f_c(x) = c + x$	$\to$	$\frac{df}{dx} = 1$

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$



[upstream gradient] x [local gradient]

w0: [0.2] x [-1] = -0.2

x0: [0.2] x [2] = 0.4

$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$

$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$

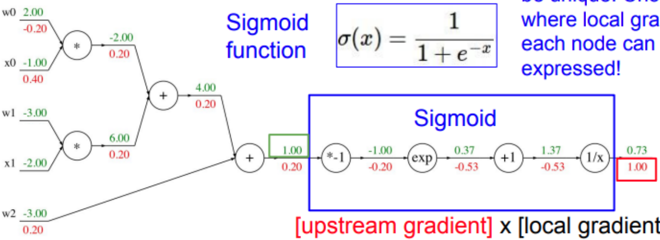
$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$

Examples

Another example: $f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

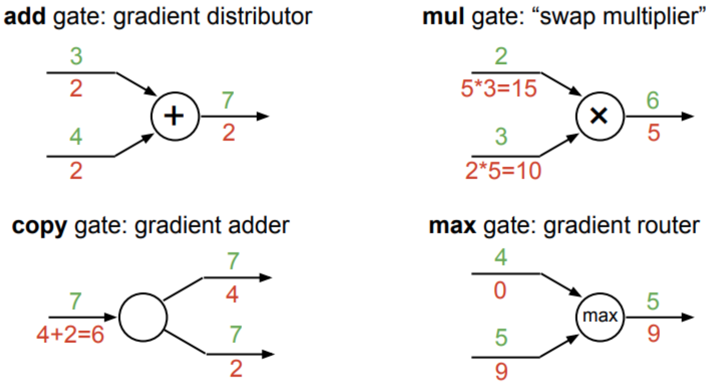


[upstream gradient] x [local gradient]

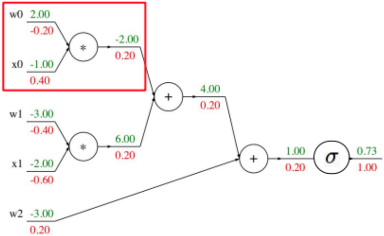
[1.00] x [(1 - 1/(1+e[1])) (1/(1+e[1]))] = 0.2

Sigmoid local gradient: $\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$

Patterns in gradient flow



Backprop Implementation: "Flat" code



Forward pass: Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)

    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

Multiply gate

Vector derivatives

Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x}\right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Vector to Vector

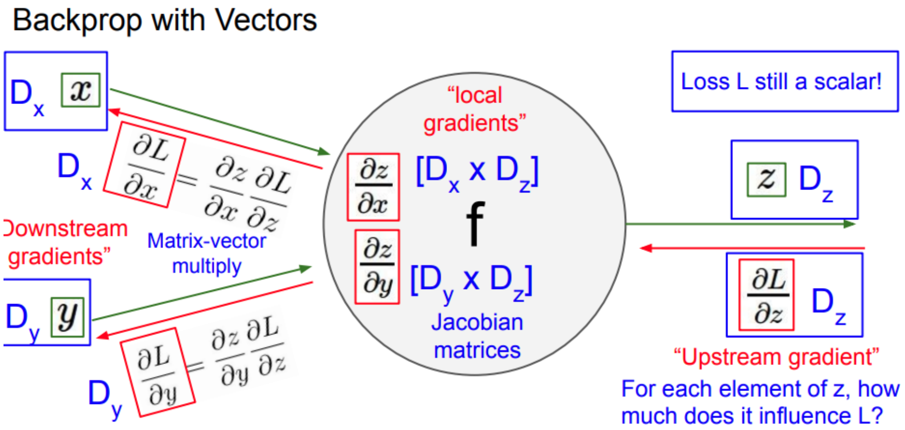
$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x}\right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

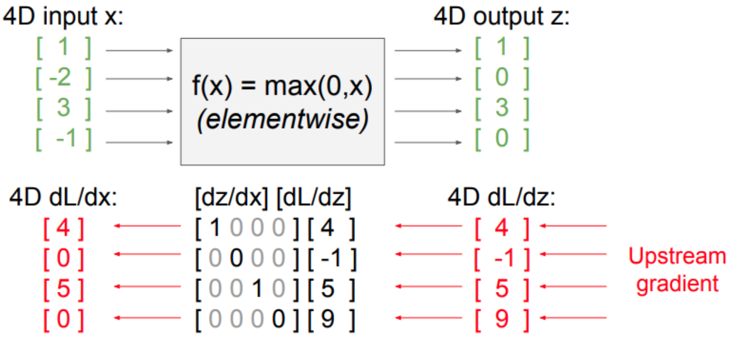
For each element of x, if it changes by a small amount then how much will each element of y change?

Vector derivatives

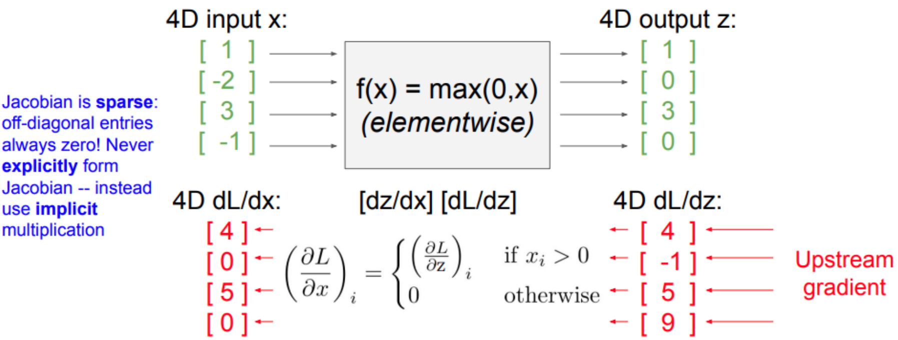


Vector derivatives: Example

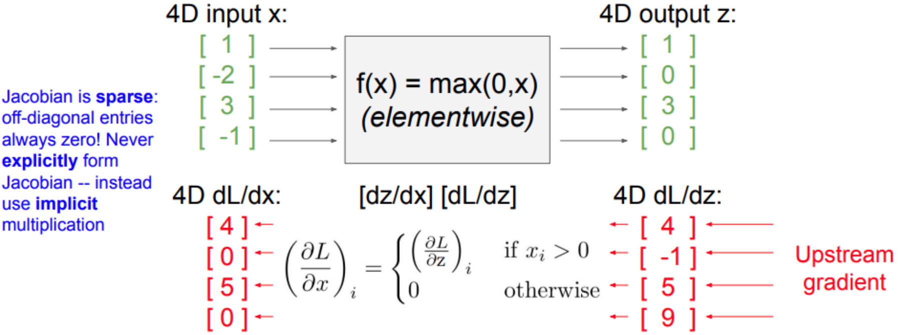
Jacobian is **sparse**: off-diagonal entries always zero! Never **explicitly** form Jacobian -- instead use **implicit** multiplication



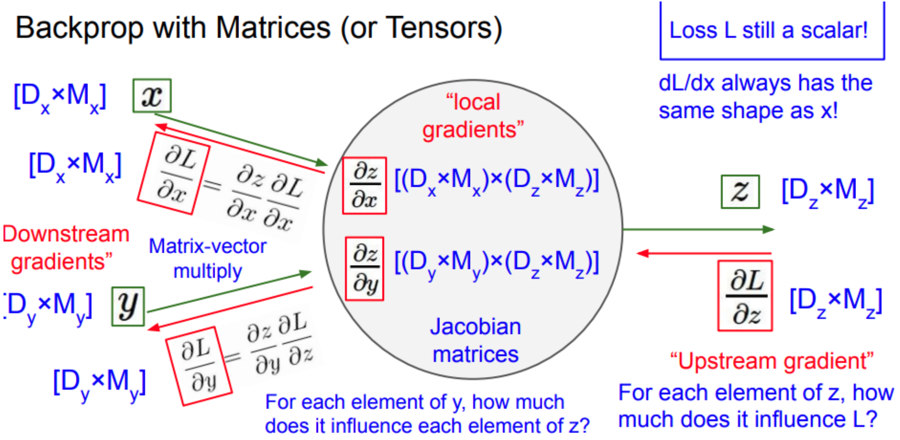
Vector derivatives: Example



Metrices derivatives: Example



Metrices derivatives: Example



Metrices derivatives: Example

Backprop with Matrices

x: [N×D]

[2 $\boxed{1}$ -3]
[-3 4 2]

w: [D×M]

[3 2 1 -1]
[2 1 $\boxed{3}$ 2]
[3 2 1 -2]

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

[13 9 $\boxed{-2}$ -6]
[5 2 17 1]

dL/dy: [N×M]

[2 3 -3 9]
[-8 1 4 6]

Q: What parts of y are affected by one element of x?
A: $\boxed{x_{n,d}}$ affects the whole row $y_{n,\cdot}$

Q: How much does $\boxed{x_{n,d}}$ affect $\boxed{y_{n,m}}$?
A: $\boxed{w_{d,m}}$

$$\frac{\partial L}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} \frac{\partial y_{n,m}}{\partial x_{n,d}} = \sum_m \frac{\partial L}{\partial y_{n,m}} w_{d,m}$$

Metrices derivatives: Example

Backprop with Matrices

x: [N×D]

$$\begin{bmatrix} 2 & \boxed{1} & -3 \\ -3 & 4 & 2 \end{bmatrix}$$

w: [D×M]

$$\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & \boxed{3} & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

y: [N×M]

$$\begin{bmatrix} \boxed{13 \quad 9 \quad \boxed{-2} \quad -6} \\ 5 \quad 2 \quad 17 \quad 1 \end{bmatrix}$$

dL/dy: [N×M]

$$\begin{bmatrix} \boxed{2 \quad 3 \quad -3 \quad 9} \\ -8 \quad 1 \quad 4 \quad 6 \end{bmatrix}$$

By similar logic:

[N×D] [N×M] [M×D]

$$\boxed{\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial y}\right) w^T}$$

[D×M] [D×N] [N×M]

$$\boxed{\frac{\partial L}{\partial w} = x^T \left(\frac{\partial L}{\partial y}\right)}$$

These formulas are easy to remember: they are the only way to make shapes match up!

Optimization and activation

Next Class..